UNIVERSITAT POLITÈCNICA DE CATALUNYA UPC)
FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

GRAU EN ENGINYERIA INFORMÀTICA
ESPECIALITAT EN TECNOLOGIES DE LA INFORMACIÓ

Interfacing Open Overlay Router with Blockchain to secure the allocation and delegation of IP prefixes

Autor:
Hamid Latif Martínez

Director:
Albert Cabellos
(Departament d'Arquitectura de Computadors)

16 de Enero de 2018

Resumen

LISP es una arquitectura cuyo objetivo es el de prevenir posibles problemas de escalabilidad y de agotamiendo de direcciones IP.
No ha sido utilizada en entornos de producción debido a la necesidad de depender de Autoridades Certificadoras para asegurar las bases de datos que utiliza.
Este proyecto pretende eliminar la dependencia de esas Autoridades Certificadoras asegurando los datos en una BlockChain.
Para hacerlo, se utilizará una BlockChain propia con un algoritmo de consenso Proof of Stake.

Resum

LISP és una arquitectura l'objectiu de la qual és el de prevenir possibles problemes d'escalabilitat i esgotament d'adreces IP.
No ha estat utilitzada en entorns de producció degut a la necessitat de dependre d'Autoritats Certificadores per assegurar les bases de dades que utilitza.
Aquest projecte pretén eliminar la dependència d'aquestes Autoritats Certificadores assegurant les dades en una BlockChain.
Per fer-ho, s'utilitzarà una BlockChain pròpia amb un algoritme de consens Proof of Stake.

Abstract

LISP is an arquitecture whose objective is to prevent possible issues with scalability and exhaustion of IP addresses.
It has not been used in production environments due to the dependence of Certificate Authorities to secure the databases it uses.
This project pretends to eliminate the dependency from these Certificate Authorities by securing the data in a BlockChain.
To do it, a custom BlockChain with a consensus algorithm will be used.

Índice

1. Contexto

Existe el temor de que la tabla de enrutamiento que gestiona un router pueda llegar a crecer tanto como para que el router tenga problemas gestionándola. Por esta razón surgió hace unos años la arquitectura LISP que, mediante el desacoplamiento de las redes IP en dos campos nuevos (explicados en la sección 2.1) permite reducir las tablas de encaminamiento y, además, tener redes overlay.

Además, para conseguir cualquier cambio en la topología de una red será necesario añadir nuevo hardware o reubicar el ya existente, por lo puede ser un problema conseguirlo. Con una red overlay esto puede ser solucionado consiguiendo mapear la topología de la red a través del software.

Estas redes presentan un problema que este proyecto pretende solucionar: gestionar los datos para mantener el estado de estas redes overlay puede ser complejo e inseguro, ya que depende de servidores centralizados que garantizan la seguridad mediante certificados digitales.

LISP utiliza certificados digitales para mantener la seguridad de las bases de datos (Mapping Servers, explicados en la sección 2.1). El proyecto busca sustituir estos certificados por la tecnología BlockChain para mantenerlos seguros y distribuidos sin la necesidad de la participación de ningún actor y para conseguir una red en la que ningún miembro tenga la posibilidad de cambiar los datos sin el consentimiento del resto de los miembros.

1.1 Redes overlay

Las redes overlay están siendo usadas en los últimos años para evitar los problemas de las redes físicas, ya que permiten esquivar las limitaciones de los despliegues existentes y mejoran la infraestructura de enrutamiento sin reemplazar el hardware ya instalado. Estas redes son útiles para un amplio conjunto de casos de uso, como por ejemplo los multicast, la ingeniería de tráfico o el networking peer-to-peer.
Además, estas redes overlay son una herramienta para hacer posibles las capacidades del Software-Defined Networking (SDN)[1].
LISP es la opción más común para crear redes overlay debido a que ofrece un estándar que permite innovación low-CAPEX en la capa de red.

Es interesante destacar OpenOverlayRouter (OOR) [1], un router con software de código abierto para desplegar redes overlay. OOR utiliza LISP para gestionar las redes overlay y permite tener una solución overlay flexible, portable y extensible via implementación de espacio de usuario. OOR está disponible para Linux, Android y OpenWrt y ha sido desarrollado por, entre otros, Jordi Paillise y Albert Cabellos-Aparicio, el estudiante de doctorado cuya tesis doctoral es este proyecto y el director del proyecto, respectivamente.

[1] http://studylib.net/doc/5382729/lisp--sdn--and-opendaylight---overview [Accedido el 21 de septiembre del 2017]

1.2 OpenOverlayRouter (OOR)

OOR es un proyecto centrado en desarrollar un router open source[2] para desplegar overlays basadas en LISP. OOR puede ejecutarse en Linux, Android o en routers con OpenWrt.
OOR soporta LISP y LISP-MN (una versión ligera para dispositivos móviles) para el intercambio del estado overlay, el protocolo NETCONF para la gestión y configuración remota y los formatos LISP y VXLAN-GPE para la encapsulación.
El focus de OOR es permitir la programabilidad de la red.
OOR ha tenido éxito, y se usa para proyectos LISP de controladores SDN como LispFlowMapping en OpenDayLight y SBI LISP en ONOS.

1.3 BlockChain

La BlockChain es una base de datos distribuida que forma, tal y como su nombre indica, una cadena de bloques. Añadir un bloque nuevo requiere de un trabajo computacionalmente difícil y una prueba de que se ha realizado este trabajo, que se guarda en el bloque para que todos los miembros que lo deseen puedan comprobar que se ha realizado correctamente (calcular el resultado del trabajo es computacionalmente difícil, pero comprobar que el resultado es correcto no lo es).
Esta tecnología se usa en las criptomonedas (como Bitcoin o Ethereum) debido a su naturaleza descentralizada y su capacidad para mantener los datos sincronizados entre todos los nodos que forman la red.

1.4 Actores implicados

Las partes implicadas en este proyecto son, principalmente, la UPC y CISCO. La participación de CISCO radica en que el origen del proyecto es una tesis doctoral de un estudiante patrocinado por la empresa, mientras que la participación de la UPC se debe a que la universidad cuenta con equipos de trabajo en la tecnología LISP y este campo es de interés para la universidad.

El objetivo principal del proyecto será el de crear un prototipo que combine LISP con BlockChain para las bases de datos, usar este prototipo para realizar pruebas de rendimiento con las que poder medir el rendimiento de esta combinación y redactar un *paper* exponiendo los resultados de estas pruebas. Por lo tanto, no se espera que el software resultante sea usado en producción, por lo que sólo será usado por los 4 miembros integrantes del grupo para realizar las pruebas.
Sin embargo sí que se espera que el *paper* tenga repercusión y sea consultado por los equipos de investigación dedicado al estudio de redes.

Tanto la UPC como CISCO serán los principales beneficiados del proyecto, pues podrán tener de primera mano tanto un prototipo de LISP con BlockChain como los resultados de las pruebas de rendimiento del mismo.

[2] La licencia de OOR es Apache 2.0.

2. Estado del arte

2.1 LISP

En la actualidad, las direcciones IP identifican tanto la localización topológica de un network attachment point como la identidad de un nodo. Sin embargo, los nodos y el enrutamiento tienen requerimientos diferentes; los sistemas de routing requieren que las direcciones sean agregables y que tengan significado topológico, mientras que los nodos requieren ser identificados independientemente de su localización actual.

LISP [2] (Locator/ID Separation Protocol) es una arquitectura cuyo origen se remonta al 2006, momento en el que durante el *Routing and Addressing Workshop* de la *Internet Architecture Board* renovó su interés[3] por el diseño de una arquitectura con routing y direccionamiento escalables y debido a la preocupación sobre la escalabilidad del sitema de routing y el agotamiento del espacio de direccionamiento IPv4 [3] .

La idea principal de LISP es la de desacoplar la semántica de las direcciones IP creando dos nuevos espacios de nombres: EIDs (End-host IDentifiers) y RLOCs (Routing LOCators). Los EIDs se usan para identificar de forma única a los nodos independientemente de su ubicación topológica y son enrutados típicamente intra-domain. Los RLOCs son asignados topológicamente a los network attachment points y son típicamente enrutados inter-domain.
Por tanto, con LISP el punto donde los nodos están conectados (el edge de Internet) y el lugar en el que el routing inter-domain ocurre (el core) pueden ser separados lógicamente por un router compatible con LISP.
LISP también introduce una base de datos para guardar y obtener los mapeos entre la identidad y la localización llamada Mapping System.

[3] http://www.rfc-editor.org/in-notes/rfc4984.txt [Accedido el 21 de septiembre del 2017]

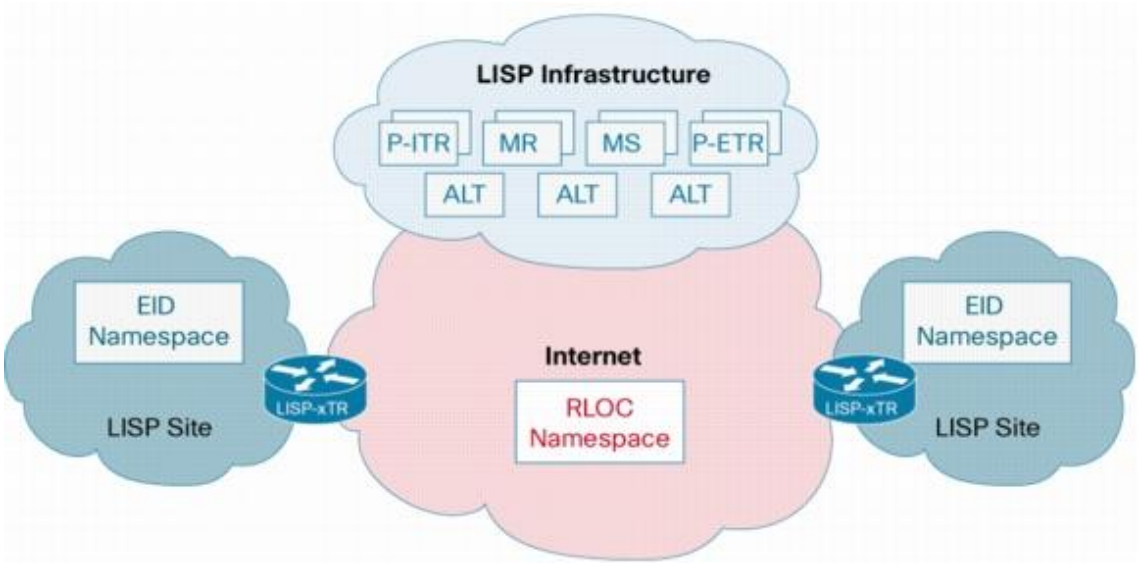


Figura 1: Entorno de despliegue de LISP

Los Mapping Server pueden formar una estructura jerárquica y distribuida: una Delegated Database Tree [4]. Con LISP-DDT se puede tener un nodo MS root cuyas direcciones RLOC apunten a un Map Server (llamado "nodo terminal DDT") o a otro nodo DDT. De esta forma se pueden ir delegando las traducciones a diversos nodos hasta llegar al nodo terminal, que contiene la traducción.
Para conseguir esta estructura se usará la tecnología de BlockChain (explicada en el apartado 2.2), lo que supondrá una novedad debido a que para implementar este árbol se suelen usar certificados.

La arquitectura LISP ha sido desarrollada a lo largo de los años, con papers como [5] describiendo detalladamente su funcionamiento para poder cumplir los estándares de la IETF o papers como [6] para estudiar diversas variables de rendimiento, por lo que se considera totalmente desarrollada y usable en un ambiente de producción.

La UPC, por ejemplo, dispone[4] de un grupo de trabajo que lleva investigando sobre LISP desde el Febrero de 2009 en colaboración con el *IP Networking Lab* de la UCL.

2.2 BlockChain

La tecnología de BlockChain [7] consiste en una lista continua de bloques en constante crecimiento en la que los enlaces se aseguran mediante una función criptográfica: el hash. Cada bloque está identificado por un hash [8] (un número fijo de bits y cuyo valor está definido por un algoritmo determinista que lo genera a partir de una entrada; en este caso, el contenido del bloque) y cada bloque contiene el hash del bloque anterior, por lo que cualquier modificación del contenido de un bloque provoca que se vea modificado su hash y que el hash que le referencia en el bloque posterior sea incorrecto, rompiendo el enlace en la cadena.

[4] https://cba.upc.edu/developed-tools/lisp-upc-tools [Accedido del 24 de septiembre del 2017]

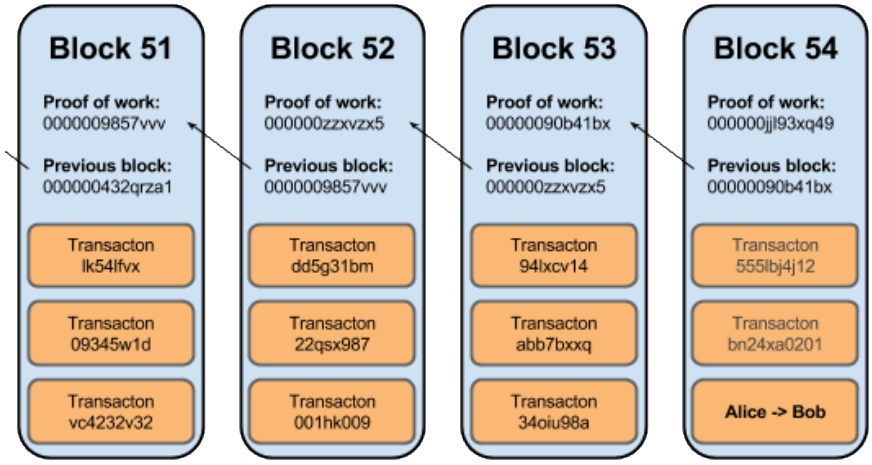


Figura 2: Cadena de bloques enlazados por la referencia al hash anterior

La idea de la tecnología BlockChain se comenta por primera vez en el paper que define Bitcoin [9], una criptomoneda que utiliza este diseño para mantener de forma segura y distribuida su base de datos.
Para evitar que modificar un bloque y recalcular el hash del mismo y todos sus sucesores no sea algo fácil computacionalmente (y, por tanto, que la seguridad se vea reducida al hecho de confiar en que el atacante sea lo suficientemente vago como para no hacerlo y no a que realmente no pueda hacerlo si quisiera) Nakamoto presenta una propuesta: el Proof-of-Work[5] (PoW) que como su nombre indica, es una prueba de que se ha debido realizar un trabajo previo a la creación del bloque.

2.2.1 Proof-of-Work

La idea tras el PoW consiste en lo siguiente: para conseguir un trabajo computacionalmente difícil se añade un campo nuevo al bloque; el "nonce". Este campo no tendrá valor y deberá ser el usuario que esté intentando crear el bloque (minándolo) el que se lo dé. El hash del bloque deberá cumplir una dificultad que vendrá dada por el número de 0s que deberán tener los n primeros bits (empezando a contar por la izquierda) del hash resultante. Un ejemplo sería:

- Hash del bloque (sin incluir el nonce): 0xFFFF
- Factor de dificultad: 2

Para resolver el PoW correctamente, se deberán asignar valores al nonce de forma incremental (empezando por un valor aleatorio) hasta que se cumpla que el hash de los contenidos del bloque (incluyendo el nonce) es igual o menor a 0x3FFF.

Este trabajo computacionalmente difícil garantiza que en caso de modificar un bloque de la cadena, se deberá recalcular el PoW de ese bloque y todos los que le suceden dando como resultado un trabajo imposible de completar por un solo usuario teniendo

[5] https://en.bitcoin.it/wiki/Proof_of_work [Accedido el 21 de septiembre del 2017]

en cuenta que el resto de los usuarios continuará la cadena original y que ésta seguirá creciendo, sumándose más trabajo del que puede resolver al atacante.

Adicionalmente, el coste de comprobar si el hash de los contenidos del bloque es menor o igual a cierto valor es sencillo computacionalmente en comparación con el hecho de tener que calcular todas las combinaciones posibles de valores del nonce, debido a que este campo suele tener un rango grande (por ejemplo, 32 bits en Bitcoin), por lo que se puede comprobar fácilmente si el resultado es correcto.

Con esta idea se garantiza que para que un ataque contra una cadena de bloques tenga éxito los atacantes han de sumar el 51% o más de la capacidad de cómputo de la red [10].

En la actualidad, la mayoría de las criptomonedas (como Bitcoin o Ethereum [11]) implementan un sistema de PoW como el descrito en el apartado 2.2.1, por lo que es algo muy estudiado y probado.

PoW tiene un lado negativo: el consumo eléctrico es muy elevado ya que la capacidad de cómputo crece constantemente.
En la actualidad, se estima que Ethereum consume más de 4.5TWh[6] por año, un valor superior incluso al de países como Siria o Jordania, y se cree que el coste energético del minado de Bitcoin puede superar a la recompensa obtenida [12].

Para solucionar esto, se ha propuesto un sistema de minado diferente al PoW, el Proof-of-Stake (PoS).

2.2.2 Proof-of-Stake

El PoS [13] supone el gran cambio que se espera que ocurra en las BlockChain debido a sus implicaciones (menor consumo energético y menor coste para empezar a minar [14]) y es la frontera del conocimiento en las tecnologías de BlockChain.

Algunas criptomonedas ya implementan versiones precoces de PoS [14] con algunas limitaciones (como el hecho de que todas las monedas posibles ya estén creadas desde un inicio y no se vayan creando monedas nuevas como, por ejemplo, en Ethereum), pero los equipos de desarrolladores de las dos grandes criptomonedas (Bitcoin y Ethereum) siguen investigando como aplicar PoS.

Los algoritmos PoS tienen como objetivo cambiar el método de minado; en vez de minar mediante capacidad de cálculo, en PoS se busca minar a través de la cantidad de moneda que se tiene respecto al total. Por ejemplo, si un usuario tiene un 1% del total podría minar el 1% de los bloques totales, ya que sería escogido el minero del bloque en 1 de cada 100 casos.

[6] https://hothardware.com/news/ethereum-and-bitcoin-energy-consumption-surpass-entire-countries-power-budgets [Accedido el 21 de septiembre del 2017]

En el caso de Ethereum los desarrolladores llevan años trabajando en un algoritmo de PoS: Casper[7].

La idea principal de Casper es la de que el usuario que quiera optar a minar un bloque tenga que depositar una cierta cantidad de monedas en depósito, que quedará retenido hasta la finalización del minado. Si el usuario intenta minar algo incorrecto (un bloque que no forma parte de la cadena original) se le penaliza retirándole los fondos depositados y no permitiéndole minar en el futuro. Si, por el contrario, mina un bloque correcto, se le devuelve el depósito más un tanto por ciento de recompensa. De esa "apuesta" (stake) viene el nombre del Proof-of-Stake.

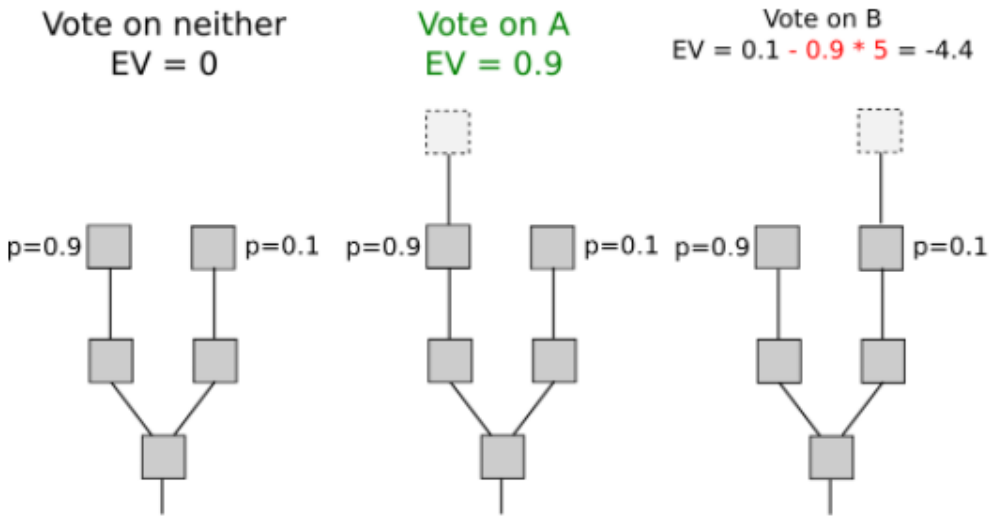


Figura 3: Ejemplo de posible recompensa apostando por la cadena "correcta" (A) y penalización apostando por la cadena "incorrecta" (B)

En la actualidad, el protocolo Casper ha sido mejorado para afrontar toda una retahíla de críticas que fueron realizadas en su origen [15]. Entre ellas, que Casper no demostraba las afirmaciones de sus autores sobre su supuesta mayor seguridad que PoW (se argumentaba que el coste del ataque sería mucho mayor que el de la defensa debido a que un ataque requeriría una gran cantidad de dinero para ser llevado a cabo).
Desde el 2014 (momento en el que fue presentado Casper) se ha seguido trabajando en desarrollar el protocolo, llegando al estado actual, cuando Casper ya se encuentra en las fases finales de desarrollo [16] (se espera que se pueda activarse en Ethereum el año 2018). Obviamente, tras su implementación quedaría ver cómo acaba funcionando, por lo que como puede observarse, se está en el límite del conocimiento en el campo en cuestión.

Para el proyecto se ha decidido utilizar PoS en lugar de PoW por diversas razones:

[7] https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/ [Accedido el 21 de septiembre del 2017]

1. Al contrario que en PoW no se necesita equipo costoso que permita tener una gran capacidad de cálculo. Esto no representa un problema en una red como Bitcoin en la que diversos usuarios participarán en una "carrera de armamento" por conseguir tener cada vez más capacidad de cálculo, pero sí en una red como la que usará esta idea, ya que en dicha red los usuarios serán empresas de internet que buscarán un coste de mantenimiento bajo.

2. En PoW la seguridad de la cadena viene dada por el crecimiento constante de la misma. Esto es así debido a las transacciones que realizarán constantemente los miles de usuarios que formarán parte de ella, pero en esta nueva red los usuarios serán muchos menos y las transacciones sólo se producirán cada vez que un nodo quiera crear un nuevo mapping de LISP, por lo que un protocolo PoW puede ser inseguro.

3. Investigar en PoS e intentar conseguir un algoritmo que lo implemente de forma adecuada a las necesidades de los usuarios de nuestra red.

3. Alcance

3.1 Objetivos

Los objetivos del proyecto son los siguientes:

- Diseñar un sistema de Base de Datos con el que poder gestionar los datos de la BlockChain.
- Crear una API compatible con OOR para poder utilizar el prototipo con las direcciones LISP ya utilizadas hoy en día, sin necesidad de cambios, y para que OOR pueda hacer peticiones de resolución al prototipo y éstas sean respondidas.
- Diseñar un sistema P2P para la comunicación entre nodos.
- Diseñar un protocolo de Proof-of-Stake que pueda ofrecer un nivel de seguridad y fiabilidad comparable al que sería necesario en un ambiente de producción.

3.2 Meta final

La meta final del proyecto es completar los objetivos previstos en el apartado 3.1 en un espacio de tiempo lo suficientemente breve como para poder realizar pruebas del rendimiento del mismo. El motivo por el que se desean hacer pruebas es que el proyecto forma parte de una tesis doctoral en la que se desea hacer un paper del resultado del proyecto, por lo que se necesitan pruebas de rendimiento que permitan demostrar si el prototipo funciona o no. Para lograrlo se empezará la fase de investigación en Agosto, de forma que en Septiembre se pueda empezar con la gestión del proyecto (separación del trabajo en paquetes para ser distribuidos entre los miembros del grupo, previsión temporal para la realización de cada paquete, etc) y la implementación lo antes posible.

Además, la UPC ya tiene varias direcciones de LISP, por lo que la posibilidad de utilizar este prototipo para gestionarlas podría dar más interés al proyecto.

Se espera tener un algoritmo de PoS que, como mínimo, ofrezca un nivel de seguridad para los datos de la BlockChain comparable a la que ofrecen los certificados digitales debido a que el prototipo busca sustituirlos y no tendría ninguna utilidad si eso pasase. Para lograr esto se usarán artículos de investigación e implementaciones de otras versiones de PoS ya existentes, de forma que se puedan estudiar sus puntos fuertes y débiles (y el motivo de los mismos) y así poder predecir mínimamente los posibles orígenes de los problemas de seguridad.

También se desea que el prototipo resultante sea lo suficientemente modular como para que las diversas partes que lo formen (gestión de la base de datos, algoritmo de PoS, diseño de las transacciones y bloques, etcétera) puedan ser reaprovechadas pudiendo reescribir los módulos que se desee. El motivo de esto es que el prototipo será de código abierto y que reaprovechar código de otras BlockChains es complejo debido a la interdependencia de las partes, por lo que sería deseable que en este caso no sea así. Para conseguir este diseño modular se separará cada una de las 4 partes principales del proyecto (API compatible con LISP, Algoritmo de consenso, Base de Datos y P2P) y se crearán APIs para la comunicación entre ellas, de forma que sea posible sustituir una de las partes y el resto puedan seguir siendo usadas fácilmente.

3.3 Posibles obstáculos

Hay múltiples posibles obstáculos que dificultarán (algunos ya lo están haciendo) la realización del proyecto.

Tal y como se ha explicado, pese a que la idea detrás del Proof-of-Stake no sea muy nueva, sus diferentes implementaciones sí que son muy novedosas (y en la mayoría de casos aún no están terminadas), por lo que el diseño de un algoritmo PoS se convierte en un objetivo muy difícil. Además, el PoS juega un papel importantísimo en la BlockChain, por lo que su funcionamiento ha de ser correcto para un gran número de casos (usuarios que conectan y desconectan de la red constantemente, usuarios que deciden no minar, usuarios que decidan minar y dejar de minar más tarde, etcétera).

La dificultad del PoS genera otra dificultad añadida. Cualquier cambio en el protocolo que pueda causar, por ejemplo, la necesidad de un atributo nuevo en la estructura de los bloques supondría, evidentemente, tener que añadir este nuevo atributo y tener que adaptar todo lo que dependa del bloque al cambio. Pese a que pueda no ser algo preocupante, indudablemente supondría una nueva dificultad a tener en cuenta.

Otro obstáculo que existe es el de la gestión del trabajo. El grupo de trabajo del proyecto está formado por cuatro estudiantes, todos ellos realizando el Trabajo de Fin de Grado, y tener un grupo así de numeroso plantea una dificultad extra a la hora de trabajar. Es notable, además, que no todos los cuatro miembros del grupo pueden acudir asiduamente a la Facultad (u otro lugar de Barcelona) debido a que no todos

residen en la ciudad, por lo que las reuniones han de ser por teleconferencia, con la dificultad añadida que eso supone.

Por último, y tal y como se ha dicho en el apartado anterior, la necesidad de tener un prototipo usable generará la dificultad de tener que realizar este proyecto en un espacio de tiempo algo ajustado para la gran cantidad de trabajo que habrá.

4. Metodología y rigor

4.1 Método de trabajo

El método de trabajo será iterativo con división del trabajo en cuatro partes repartidas entre los miembros del grupo.
Las cuatro partes serán:

- Repositorio de datos: incluirá la gestión de la base de datos, la estructura del bloque y las transacciones, los forks de la cadena, las transacciones pendientes y el estado actual de la cadena.
- Algoritmo de consenso: incluirá el algoritmo PoS.
- Validación y P2P: incluirá la gestión del protocolo P2P de la red, la gestión de los forks (decidir qué cadena escoger en caso de fork) y la validación de los datos.
- API compatible con LISP: incluirá la gestión de las direcciones en formato LISP, los nuevos mapeos de direcciones LISP (asignarle un Mapping Server a una traducción) y la comunicación entre OOR y el prototipo.

Cada una de estas partes recaerá sobre un miembro del grupo, que será el principal encargado de desarrollarla y de documentarla. En concreto, yo realizaré la conexión entre OOR y el prototipo mediante la API compatible con LISP.

Habrá una reunión semanal (realizada a través de videoconferencia) para poner en común los avances que se vayan produciendo en cada uno de los bloques y poder comentar y solucionar los posibles problemas que vayan surgiendo a lo largo del desarrollo.

4.2 Herramientas de seguimiento

Las herramientas usadas para el seguimiento serán Github (que se usará para tener un directorio común con todo el código del prototipo), Skype (usado para las reuniones semanales) y una IDE de desarrollo a decidir por cada miembro del equipo (en el caso del autor de este informe, PyCharm) y email (para poder contactar entre los miembros en caso de necesidad).

También se dispone de un aula (D6-108) para poder tener reuniones presenciales en caso de ser necesario (y posible).

4.3 Métodos de validación

Se usarán clases de testeo para validar las distintas partes del trabajo y comprobar que los módulos que se vayan programando funcionan.
Se empezará por las clases más básicas (el bloque o la transacción, por ejemplo) para dejar para el final las pruebas del algoritmo de consenso (PoS) con la previsión de que será el que más problemas dé.

5. Descripción de las tareas

5.1 Duración del proyecto

El proyecto será desarrollado entre Julio de 2017 y Enero de 2018, ambos incluidos.
La inicialización del proyecto tendrá lugar en Julio, mes en el que se realizará la investigación necesaria para poder realizarlo.
Tras esto, se empezará con el desarrollo del proyecto en Septiembre y se seguirá hasta Diciembre.
En Enero tendrá lugar la documentación del proyecto y la presentación.

5.2 Tareas

El trabajo estará dividido en tareas comunes (como la investigación o la documentación) y en tareas individuales (como cada uno de los 4 bloques de desarrollo, que serán realizados por un miembro del grupo).

Tarea	Responsable	Horas
Inicio del proyecto		55
1. Búsqueda y puesta en marcha	Jefe de Proyecto	5
2. Investigación	Desarrolladores	50
Gestión del proyecto		50
1. Definición del alcance	Jefe de Proyecto	10
2. Planificación temporal	Jefe de Proyecto	5
3. Gestión económica y sostenibilidad	Jefe de Proyecto	5
4. Presentación preliminar	Jefe de Proyecto	10
5. Pliego de condiciones	Jefe de Proyecto	5
6. Presentación y documento final	Jefe de Proyecto	15
Implementación		560
LISP		140
1. Análisis de requisitos	Desarrollador (Hamid)	10
2. Diseño	Desarrollador (Hamid)	20
3. Scripts de control	Desarrollador (Hamid)	40
4. Integración BlockChain	Desarrollador (Hamid)	40
5. Creación API	Desarrollador (Hamid)	30
Gestión de Datos		140
1. Análisis de requisitos	Desarrollador (Miquel)	10
2. Diseño	Desarrollador (Miquel)	20
3. Estructuras de Datos	Desarrollador (Miquel)	35
4. Cadena de Bloques	Desarrollador (Miquel)	30

5. Validación	Desarrollador (Miquel)	15
6. Creación API	Desarrollador (Miquel)	30
Algoritmo de Consenso		140
1. Análisis de requisitos	Desarrollador (Eric)	10
2. Diseño	Desarrollador (Eric)	20
3. Implementación Algoritmo	Desarrollador (Eric)	80
4. Creación API	Desarrollador (Eric)	30
P2P + Protocolo		140
1. Análisis de requisitos	Desarrollador (Carlos)	10
2. Diseño	Desarrollador (Carlos)	20
3. Estructura de la Transacción	Desarrollador (Carlos)	20
4. Red P2P	Desarrollador (Carlos)	35
5. Resolución de Forks	Desarrollador (Carlos)	25
6. Creación API	Desarrollador (Carlos)	30
Testing		30
1. Creación de tests	Desarrolladores	20
2. Ejecución de tests	Desarrolladores	10
Mediciones		25
1. Definición	Ingeniero QA	5
2. Implementación	Desarrolladores	10
3. Ejecución	Ingeniero QA	10
Documentación		60
1. Redacción	Jefe de Proyecto	50
2. Preparación defensa	Jefe de Proyecto	10

Tabla 1: Duración de las tareas

5.3 Dependencias

En el Gantt de la página siguiente pueden verse las dependencias que dividen el proyecto en 3 fases:

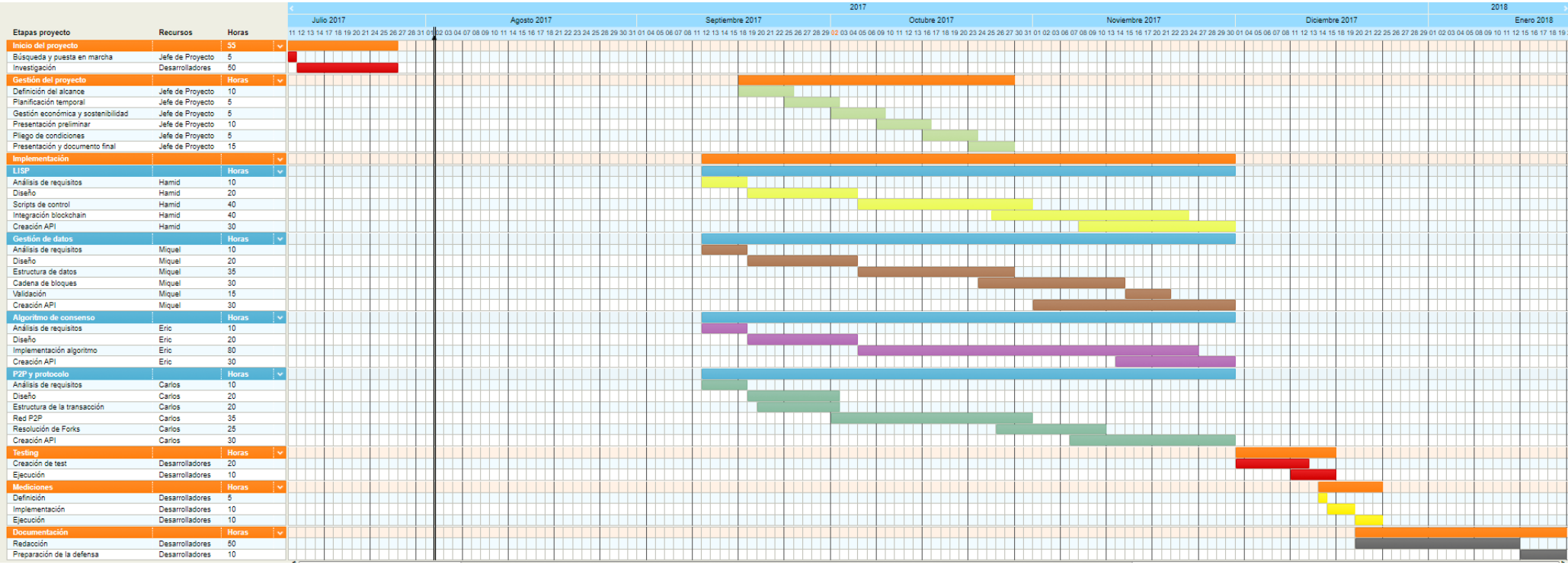
1. Fase de investigación
2. Fase de implementación
3. Fase de testeo y documentación

La primera fase tiene lugar en Julio. Durante este mes el Jefe de Proyecto presentará la idea del proyecto al resto el grupo, explicará los objetivos principales del mismo y explicará y dividirá la carga de trabajo entre los miembros.
Tras esto, cada responsable de su bloque ha de iniciar la investigación, que consistirá en documentarse sobre las herramientas que necesitará para resolver su parte y en leer sobre el estado del arte del tema y ejemplos existentes.

La segunda fase del proyecto se inicia el 12 de Septiembre y dura hasta el 30 de Noviembre.
Durante ese periodo el grupo en conjunto ha de trabajar en la gestión del proyecto y cada miembro en su bloque.
Para la realización de cada bloque se empezará con un análisis de requisitos y un diseño con el que poder dar forma al bloque e identificar todas las partes necesarias.

Tras esto se inicia con el desarrollo de cada una de esas partes, empezando por cada una de las partes necesarias del bloque y acabando por la creación de la API para que cada bloque pueda ser usado en combinación con el resto.
Finalizada la implementación

La tercera fase del proyecto se inicia el 1 de Diciembre y acaba el 19 de Enero. Durante ese periodo el grupo ha de trabajar en la creación de un conjunto de pruebas y usarlas para medir diversas métricas. Tras esto se iniciará la documentación del proyecto y la preparación de la defensa delante del tribunal.



Etapas proyecto	Recursos	Horas
Inicio del proyecto		35
Búsqueda y puesta en marcha	Jefe de Proyecto	5
Investigación	Desarrolladores	50
Gestión del proyecto		Horas
Definición del alcance	Jefe de Proyecto	10
Planificación temporal	Jefe de Proyecto	5
Gestión económica y sostenibilidad	Jefe de Proyecto	5
Presentación preliminar	Jefe de Proyecto	10
Pliego de condiciones	Jefe de Proyecto	5
Presentación y documento final	Jefe de Proyecto	15
Implementación		
LISP		Horas
Análisis de requisitos	Hamid	10
Diseño	Hamid	20
Scripts de control	Hamid	40
Integración blockchain	Hamid	40
Creación API	Hamid	30
Gestión de datos		Horas
Análisis de requisitos	Miquel	10
Diseño	Miquel	20
Estructura de datos	Miquel	35
Cadena de bloques	Miquel	30
Validación	Miquel	15
Creación API	Miquel	30
Algoritmo de consenso		Horas
Análisis de requisitos	Eric	10
Diseño	Eric	20
Implementación algoritmo	Eric	80
Creación API	Eric	30
P2P y protocolo		Horas
Análisis de requisitos	Carlos	10
Diseño	Carlos	20
Estructura de la transacción	Carlos	20
Red P2P	Carlos	35
Resolución de Forks	Carlos	25
Creación API	Carlos	30
Testing		Horas
Creación de test	Desarrolladores	20
Ejecución	Desarrolladores	10
Mediciones		Horas
Definición	Desarrolladores	5
Implementación	Desarrolladores	10
Ejecución	Desarrolladores	10
Documentación		Horas
Redacción	Desarrolladores	50
Preparación de la defensa	Desarrolladores	10

Tabla 2: Diagrama de Gantt

5.4 Recursos

Para la realización del proyecto será necesario un conjunto de recursos que comprenderán desde los ordenadores usados para la implementación a programas informáticos necesarios para la comunicación. Por tanto, se consumirán los siguientes recursos hardware y software:

Recurso	Uso
Ordenadores (uno por miembro)	Ordenador personal desde el que se implementarán cada uno de los bloques
Skype	Comunicación entre los miembros del grupo
Google drive	Compartición de documentos entre los miembros del grupo
Tomsplanner	Realización del diagrama de Gantt
Git	Control de versiones
Github	Repositorio de código
Word	Escritura de la documentación
Python	Lenguaje en el que se escribirá el proyecto
Pyethereum	Se intentará reaprovechar código de la implementación de Ethereum en Python
Email	Comunicación entre los miebros del grupo

Tabla 3: Recursos software y hardware necesarios

Los recursos humanos necesarios serán los siguientes:

Rol	*Horas*
Jefe de proyecto	*115*
Desarrolladores (total)	*650*
Ingeniero QA	*15*

Tabla 4: Recursos humanos

6. Valoración de alternativas y acción ante las mismas

6.1 Riesgos

La realización del proyecto conllevará un considerable número de riesgos que habrá que tener en cuenta a la hora de esperar posibles contratiempos. En concreto, preocupan los riesgos de la fase de implementación:

En el bloque LISP es vital conseguir la usabilidad de la API con las direcciones LISP que actualmente posee la UPC. Por tanto, que la implementación se ajuste estrictamente a

los estándares LISP es un objetivo básico y cualquier posible desviación un riesgo a tener en cuenta.

En el bloque de gestión de datos es muy importante conseguir buenas estructuras de datos que permitan tener todos los datos necesarios pero sin tener redundancias, además de encadenar los bloques de forma correcta, pudiendo obtener los datos de cualquier bloque de forma rápida. Un riesgo que se tiene en cuenta es no conseguir un diseño óptimo y tener problemas a la hora de su uso, por lo que es muy importante prestar mucha atención en este punto, ya que cualquier arreglo podría suponer un gran coste en cuanto a tiempo.

En el bloque del algoritmo de consenso el gran problema (valga la redundancia) es el algoritmo de consenso, y un riesgo que se tiene en cuenta es el de conseguir un algoritmo poco seguro, algo que podría echar por tierra una de las funcionalidades principales del proyecto (seguridad de los datos) y que supondría una gran inversión de tiempo para arreglar. Este es, quizás, el mayor riesgo del proyecto.

En el bloque del P2P un gran riesgo es el de no conseguir resolver todos los forks de forma correcta, por lo que también habrá que tener muy en cuenta este punto, ya que como en el caso del algoritmo de consenso, su correcto funcionamiento es vital para el proyecto.

El consumo de recursos no se verá afectado a causa de un posible contratiempo debido a que no será necesario ningún recurso extra (entendiendo que un uso mayor de horas de ordenadores u otro hardware o software no supone un consumo extra de recursos).

6.2 Alternativas y acciones

Sabiendo los riesgos a los que se expone el proyecto se pueden idear acciones para prevenir estos riesgos y alternativas para evitar que la materialización de algún riesgo afecte mucho a la previsión temporal.

La prevención empezará en la fase de investigación ya que se estudiará el funcionamiento de Ethereum para intentar entender el por qué de las decisiones que han tomado sus desarrolladores e imitarlas (adaptando el código de la implementación de Ethereum en Python cuando sea posible), esperando que imitándolas se reduzca el riesgo de que alguno de esos fallos se produzca.

También se analizarán los algoritmos de consenso PoS propuestos para poder tener una visión general de los problemas y las críticas que están recibiendo para intentar poner solución a estos y analizar si suponen un problema en el uso esperado del prototipo.

Adicionalmente se idearán planes alternativos en caso de que la prevención falle.

Si el algoritmo de consenso o la validación de los forks dan problemas se ampliará el tiempo de desarrollo, reduciéndolo del tiempo de pruebas, ya que se prefiere tener menos pruebas de un diseño completamente funcional que pruebas de uno con fallos.

En el caso de que los riesgos “menores” (los relacionados con LISP y con las estructuras de datos) puedan dar lugar a problemas se asignará un compañero extra para ayudar en el desarrollo de ese bloque y evitar que el problema ralentice el desarrollo normal del mismo.

Por tanto, se espera que la prevención y los planes alternativos consigan que la única modificación temporal necesaria pueda ser una reducción de las horas dedicadas a las pruebas, consiguiendo cumplir el objetivo principal del proyecto, que es un prototipo completamente funcional.

7. Identificación de los costes

El proyecto genera unos costes cuyos orígenes se dividen en recursos humanos y en recursos materiales (tanto costes directos como indirectos). Se procederá a calcular una estimación en las siguientes secciones.

7.1 Costes en recursos humanos

El proyecto será desarrollado por 4 personas (cada una de ellas es un alumno de la FIB cursando el TFG). A continuación se define el precio por hora de cada rol:

Rol	Salario
Programador	10€/h
Jefe de proyecto	20€/h
Ingeniero de QA	15€/h

Tabla 5: Asignación de salarios

A partir de esta asignación se calcula el coste total en recursos humanos del proyecto:

Tarea	Responsable	Horas	Coste
Inicio del proyecto		55	
1. Búsqueda y puesta en marcha	Jefe de Proyecto	5	100€
2. Investigación	Desarrolladores	50	500€
Gestión del proyecto		50	
1. Definición del alcance	Jefe de Proyecto	10	200€
2. Planificación temporal	Jefe de Proyecto	5	100€
3. Gestión económica y sostenibilidad	Jefe de Proyecto	5	100€
4. Presentación preliminar	Jefe de Proyecto	10	200€
5. Pliego de condiciones	Jefe de Proyecto	5	100€
6. Presentación y documento final	Jefe de Proyecto	15	300€
Implementación		560	
LISP		140	

1. Análisis de requisitos	Desarrollador (Hamid)	10	100€
2. Diseño	Desarrollador (Hamid)	20	200€
3. Scripts de control	Desarrollador (Hamid)	40	400€
4. Integración BlockChain	Desarrollador (Hamid)	40	400€
5. Creación API	Desarrollador (Hamid)	30	300€
Gestión de Datos		140	
1. Análisis de requisitos	Desarrollador (Miquel)	10	100€
2. Diseño	Desarrollador (Miquel)	20	200€
3. Estructuras de Datos	Desarrollador (Miquel)	35	350€
4. Cadena de Bloques	Desarrollador (Miquel)	30	300€
5. Validación	Desarrollador (Miquel)	15	150€
6. Creación API	Desarrollador (Miquel)	30	300€
Algoritmo de Consenso		140	
1. Análisis de requisitos	Desarrollador (Eric)	10	100€
2. Diseño	Desarrollador (Eric)	20	200€
3. Implementación Algoritmo	Desarrollador (Eric)	80	800€
4. Creación API	Desarrollador (Eric)	30	300€
P2P + Protocolo		140	
1. Análisis de requisitos	Desarrollador (Carlos)	10	100€
2. Diseño	Desarrollador (Carlos)	20	200€
3. Estructura de la Transacción	Desarrollador (Carlos)	20	200€
4. Red P2P	Desarrollador (Carlos)	35	350€
5. Resolución de Forks	Desarrollador (Carlos)	25	250€
6. Creación API	Desarrollador (Carlos)	30	300€
Testing		30	
1. Creación de tests	Desarrolladores	20	200€
2. Ejecución de tests	Desarrolladores	10	100€
Mediciones		25	
1. Definición	Ingeniero QA	5	75€
2. Implementación	Desarrolladores	10	100€
3. Ejecución	Ingeniero QA	10	150€
Documentación		60	
1. Redacción	Jefe de Proyecto	50	1000€
2. Preparación defensa	Jefe de Proyecto	10	200€
			9025€

Tabla 6: Cálculo de costes de recursos humanos

Cabe destacar que, pese a que los miembros del grupo no podrán reunirse presencialmente pues no todos residen en zonas cercanas a Barcelona, las reuniones podrán realizarse vía videoconferencia, con coste nulo, por lo que no se considerarán costes de movilidad.

7.2 Costes en recursos materiales

7.2.1 Costes directos

- **Equipo personal:** Un elemento a tener en cuenta en el coste de recursos materiales es la amortización del equipamiento personal. Suponiendo un coste por equipamiento de 1500€, que se necesitan 4 (uno para cada miembro del grupo), una vida útil de 4 años, que un año tiene 260 días laborales y que cada

día laboral tiene 8 horas, el coste/h de la amortización son: 6000€/8320h = 0.72€/h. Con un uso necesario de 780h, el coste total de la amortización es de 561.6€.

- **Herramientas de desarrollo y extras:** Las herramientas usadas tienen una licencia de software open source con coste gratuito, por lo que no supondrán coste alguno para la realización del proyecto. Adicionalmente, la UPC ya cuenta con una red beta de LISP, por lo que el coste de la misma será 0.

7.2.2 Costes indirectos

- **Consumo eléctrico:** Se supondrá que el consumo eléctrico por hora de cada uno de los equipos será de 0.5€, por lo que el coste económico de 780h será de 390€.
- **Acceso a Internet y oficinas:** El coste del acceso a Internet será de 45€/mes, por lo que, suponiendo que se realiza entre los meses de Julio y Enero (sin contar Agosto), el gasto para los 4 miembros será de 6meses*45euros/mes*4personas = 1080€. Las oficinas no supondrán ningún coste, ya que el proyecto se desarrollará en la universidad o en casa.
- **Contingencias:** La contingencia cubrirá cualquier posible desvío en los costes del proyecto y se fijará en el 20% del coste total.
- **Imprevistos:** Tal y como se comentó en la entrega anterior, cualquier posible retraso en la finalización del desarrollo será cubierto con una menor dedicación a la creación de tests, por lo que se supondrá que debido a que tanto el desarrollo de cada paquete (LISP, Gestión de datos, etc.) como del testing corre a cargo de desarrolladores, no supondrá ningún coste extra dedicar más tiempo a desarrollo y menos a testing.

7.3 Resumen de costes

RRHH	Equipo personal	H. desarrollo	Consumo eléctrico	Acceso I. y ofic.	Contingencias	Imprevistos	Total
9025€	561.6€	0€	390€	1080€	2211.32€	0€	13267.92€

Tabla 7: Resumen de costes totales

Cada uno de los costes de la tabla 7 se obtiene de las secciones 7.1.1 y 7.1.2, en las que se estima el coste aproximado de cada uno de los términos.

8. Control de gestión

Para controlar los costes se empezará por la identificación de riesgos, en la que se identificarán los principales riesgos que podrían causar un aumento de los mismos. Sin embargo, tal y como se ha dicho, en este proyecto no sucederá tal caso, ya que un retraso en el desarrollo de uno de los paquetes de trabajo causará que se reduzca el tiempo dedicado a los testeos, cosa que no provocará ningún aumento de costes. Pese a esto se desarrollará un estudio de riesgos, pues otro tipo de riesgos sí pueden aumentar el coste:

Elemento de riesgo	Probabilidad de ocurrencia	Impacto (0-10)	Peso
Retraso en el desarrollo de uno de los 4 bloques principales	20%	2	0.4
Fallo en equipos informáticos	5%	5	0.25

Tabla 8: Estudio de riesgos

Como puede deducirse, el fallo en los equipos informáticos sí podría causar un aumento del coste económico. Suponiendo un fallo en un único equipo (pues un fallo en dos a la vez tiene una probabilidad del 0.25%) habría que comprar un equipo nuevo (no se supone amortización), algo que tendría un coste de 1500€. Este precio entraría dentro de lo presupuestado para contingencias (2211.32€ presupuestados), por lo que el coste del proyecto no se vería afectado.

Pueden producirse desvíos en los diversos costes del proyecto, que se podrán calcular con las siguientes fórmulas:

1. Desvío de recursos humanos = (coste estimado - coste real) * horas reales
2. Desvío de consumo eléctrico = (coste estimado – coste real) * horas reales
3. Desvío de acceso a Internet y oficinas = (coste estimado – coste real) * meses reales
4. Desvío total en recursos = coste total estimado en recursos – coste total de recursos

9. Sostenibilidad

	PPP	Vida útil	Riesgos
Ambiental	Consumo del diseño	Huella ecológica	Riesgos ambientales
	8/10	12/20	-5/-20
Económico	Factura	Plan de viabilidad	Riesgos económicos
	9/10	16/20	-3/-20
Social	Impacto personal	Impacto social	Riesgos sociales
	3/10	2/20	-0/-20
Rango	20/30	30/60	-8/-60
Sostenibilidad	42/90		

Tabla 9: Matriz de sostenibilidad

9.1 Estudio del Impacto Ambiental

Durante las diferentes fases del proyecto se necesitarán diversos recursos hardware (equipos informáticos y servidores) y diversos recursos software (github, PyCharm, etc.). Los únicos recursos que generarán un consumo serán los hardware y los

humanos, que consumirán electricidad: los equipos informáticos con los que se escribirá el código (y las personas que lo escriban) lo harán durante la implementación y los servidores en los que se pruebe el mismo lo harán durante la fase de pruebas del prototipo.
Debido a que el resultado final será un prototipo, el consumo eléctrico del mismo desde su puesta en marcha hasta el fin de su vida útil será muy bajo, ya que se espera que la misma no supere el mes.
Suponiendo que una persona consume unos 0,1kWh realizando una rutina habitual (en este caso, escribiendo código), el coste eléctrico de los recursos humanos será de 780h*0,1kWh = 78kWh. El coste eléctrico de los equipos informáticos, suponiendo que consumen 1kWh, será de 780h*1kWh = 780kWh. El coste eléctrico de los servidores donde se probarán es desconocido aún, pues a falta de diseñar los tests se desconoce el número de equipos que se necesitarán ni las horas que se usarán.
Por tanto, el coste eléctrico total de la realización del proyecto será de 858kWh

Origen del consumo	Consumo
Equipos informáticos	780kWh
Recursos humanos	78kWh
Total	858kWh

Tabla 10: Desglose del consumo eléctrico por su origen

El objetivo del proyecto es añadir seguridad extra al funcionamiento de LISP mediante la tecnología BlockChain, por lo que el consumo eléctrico del mismo sin la propuesta será muy similar, ya que mantener los nodos de la BlockChain supondría ejecutar únicamente un programa con un coste computacional muy bajo en el mismo servidor LISP.

Para la realización del proyecto se ha buscado reutilizar código de la implementación en Python de Ethereum (PyEthereum) con la intención de conseguir un proyecto bien diseñado (se supone que el diseño de PyEthereum es correcto y optimizado, por lo que imitarlo hará que el prototipo también lo sea), además de diseñar una API que permita el uso de las direcciones de LISP de las que dispone la UPC sin necesidad de adaptarlas.

El prototipo será un programa software, por lo que no tendrá coste alguno utilizarlo, más allá del coste eléctrico que pueda tener mantener el servidor en el que se use, y una vez acabados los test el programa podrá eliminarse fácilmente de los servidores en los que se haya utilizado, pudiendo dejarlos en su estado inicial rápidamente y sin coste adicional.

La única contaminación generada por la realización del proyecto procederá del consumo eléctrico explicado anteriormente, ya que no requerirá de ningún componente especialmente fabricado para el mismo, y esta electricidad será la única materia prima necesaria. Adicionalmente, puede tenerse en cuenta el consumo de papel originado por el proyecto (impresión de la documentación, etc), aunque este será mínimo ya que se pretende usar Google drive para compartir documentación, evitando imprimirla.

La huella ecológica del servicio de direcciones LISP de la UPC será el mismo si se usa el prototipo o no, debido a que el prototipo ofrecerá una forma diferente de garantizar la seguridad (BlockChain en vez de certificados), pero el coste eléctrico generado por los servidores será el mismo.

La implementación del prototipo, tal y como se ha comentado en entregas anteriores, pretende ser una que debido a su documentación extensiva y a su modularidad pueda ser reutilizada para diseñar otras BlockChains. Por tanto, se pretende que se puedan reutilizarse las partes que se deseen.

El riesgo ambiental se ha valorado en -5 porque no se considera que puedan haber grandes riesgos ambientales en un proyecto software.

9.2 Estudio del Impacto Económico

Se han estimado los costes económicos que supone el proyecto (Tabla 7), que incluyen los costes de que posibles riesgos tengan lugar. No se han incluido costes adicionales que cubran la vida útil, ya que como se ha dicho se pretende crear un proyecto que será usado únicamente para probar su rendimiento y no para desplegarlo en un uso real.
El coste del proyecto se supone competitivo ya que el 68% del mismo tiene su origen en RRHH, y se entiende que estos serán los mismos aunque el proyecto sea realizado por otro equipo. Por tanto, no se puede reducir el coste del proyecto de forma significativa (se podría reducir el coste de la compra de equipos informáticos, pero se supone, de nuevo, que el coste de los mismos será similar si otro equipo realiza el prototipo).

Se cree que el proyecto no se podría realizar en menos tiempo ya que pese a que se reutilice código implementar una BlockChain y, sobretodo, idear un algoritmo de Proof-of-Stake funcional no es algo trivial y requiere de una cantidad de horas de trabajo similar a las que se han previsto para este proyecto. Este tiempo se ha dividido según la importancia y dificultad que tiene cada parte.

No se prevé colaboración alguna con otros proyectos.

El riesgo económico se ha valorado en -3 porque no se considera que pueda tener un gran impacto económico. Este prototipo aún está lejos de ser implementado en un escenario de producción y no se pueden saber con certeza las consecuencias de su aplicación en tal escenario. Se cree que no afectaría al modelo de negocio de las grandes empresas de Internet y, como mucho, podría tener algún impacto económico en las empresas certificadoras (pese a que se desconoce cuál podría ser su alcance).

9.3 Estudio del Impacto Social

Previamente a hacer este proyecto ya había trabajado con tecnologías BlockChain (en concreto, con Ethereum) y los conceptos explicados en la primera entrega (Proof-of-Work, Proof-of-Stake, etc) ya eran conocidos para mi, por lo que este proyecto no me

ha dado una nueva visión sobre, por ejemplo, el ahorro energético que podría suponer una BlockChain con un algoritmo PoS en vez de PoW.
Sí que ha sido una novedad descubrir LISP y su funcionamiento pero, de nuevo, este proyecto no generaría una mejora de la huella ecológica de su uso ni reduciría las desigualdades entre sus usuarios, por lo que no supondría una mejora en cuanto a un aspecto social.

Las grandes compañías de Internet tienen motivos (impagos de licencias que obliguen a retirar la posesión de una dirección IP, por ejemplo) para asegurar el uso de direcciones LISP mediante certificados, por lo que no queda muy claro que una opción completamente descentralizada (en la que además perderían todo el poder que ahora tienen) sea algo necesario. Este proyecto tiene como origen un documento de investigación sobre el tema, por lo que su interés es más académico que práctico (aplicarlo en un ambiente de producción).

La mayoría de consumidores no tienen conocimiento del funcionamiento de Internet ni de las implicaciones que el uso de este prototipo implicaría, y el servicio seguiría siendo el mismo, por lo que no representaría un cambio en su calidad de vida ni en su forma de usar el servicio. Sí que podría ser de interés para gente interesada en el tema (investigadores de redes, por ejemplo), pero en ningún caso mejoraría su calidad de vida.

Las grandes compañías de Internet podrían verse afectadas si se utiliza LISP con este prototipo ya que perderían todo el control que poseen sobre el mismo y pasaría a ser totalmente descentralizado y autónomo en su funcionamiento.

Se ha valorado el resgio social con un -0 debido a que no se cree que afecte en manera alguna a la sociedad. La gente “común” (la no interesada en el funcionamiento de Internet) no tendría ni idea de que un prototipo así está funcionando en un ambiente de producción en caso de que eso pasase.

10. Descripción del trabajo realizado

El trabajo, como se ha explicado, se divide en 4 partes, una por cada integrante del equipo.
En concreto, a mí me ha tocado realizar la parte de la comunicación entre OOR y el prototipo. Además, también he realizado el main del programa, que enlaza todos los módulos necesarios y los conecta para formar el proyecto.

10.1 Requisitos

Para entender el trabajo realizado, primero hay que entender por qué se ha hecho, por lo que hay que analizar los requisitos; en concreto, los relacionados con mi parte: la comunicación entre el protitpo y OOR.

El primer requisito es que la comunicación entre OOR y el prototipo deberá realizarse mediante dos sockets UDP que utilicen los puertos 16001 (para enviar desde OOR) y 16002 (para leer desde OOR).

Las peticiones que OOR enviará al prototipo tendrán siempre un mismo formato:

1. Nonce: 64 bits
2. AFI: 16 bits
3. Dirección: 32 o 128 bits, dependiendo de la AFI

Por tanto, es necesario que el prototipo sea capaz de leer estas peticiones, siendo éste el segundo requisito.

El prototipo debe ser capaz de enviar sus respuestas a OOR utilizando el siguiente formato:

1. Nonce: 64 bits
2. Flag: 8 bits que indican el contenido de la información
3. Información: una lista de map servers o un mapping

En el caso de que la información sea una lista de map servers, deberá tener el formato siguiente:

1. Counter: 8 bits que indican el número de map servers
2. AFI i: 16 bits
3. Dirección i: 32 o 128 bits dependiendo de la AFI i

En el caso de que la información sea un map reply, la respuesta tendrá el siguiente formato, que sigue el estándar definido en la sección 6.1.4 del RFC 6380[8]:

1. TTL: un entero de 32 bits que indica el tiempo, en minutos, que el receptor del map reply guardará la traducción en cache
2. Locator count: un entero de 8 bits que indica el número de locators que contiene el map reply
3. EID mask-len: un entero de 8 bits que indica la longitud de la máscara del EID-prefix
4. ACT: un entero de 3 bits que define acciones del map reply. En concreto, puede haber 4 acciones:
   4.1. No-Action: el map-cache se mantiene vivo y no ocurre encapsulación
   4.2. Natively-Forward: el paquete es natively-forwarded
   4.3. Send-Map-Request: el paquete invoca enviar un Map-Request
   4.4. Drop: un paquete que empareja esta entrada map-cache se dropea. Un mensaje ICMP Destination Unreachable debería ser enviado.
5. Authoritative: un bit que siempre se pone a 1 cuando es enviado por un ETR. Cuando un Map-Server está proxy Map-Replying el bit se pone a 0.
6. Map-Version number: un entero de 12 bits que, cuando es diferente de 0, sirve para que el Map-Reply sender informe al ITR de su número de versión para el record EID del mensaje. Cuando éste vale 0, no hay información de la versión.
7. EID-prefix-AFI: un entero de 16 bits que indica el AFI del EID

[8] https://datatracker.ietf.org/doc/rfc6830/

8. EID-prefix: un valor de 32 o 128 bits que indica la @IP y cuyo tamaño depende del AFI
9. Priority: un entero de 8 bits que indica la prioridad del locator. Un valor más pequeño indica más prioridad
10. Weight: un entero de 8 bits que indica cómo balancear el tráfico entre RLOCs con el mismo nivel de prioridad
11. M Priority: un entero de 8 bits que define la prioridad multicast del RLOC
12. M Weight: un entero de 8 bits que indica cómo balancear tráfico multicast entre RLOCs con el mismo nivel de prioridad
13. Unused Flags: un valor de 13 bits que se dejan a 0 y se ignoran
14. L: un bit que, cuando se activa, marca el Locator como un local Locator al ETR.
15. p: un bit que, cuando se activa, provoca que el ETR informe al RLOC-probing ITR que la dirección del locator para el que este bit se activa es el que se está RLOC-probing
16. R: un bit que se activa cuando el nodo que envía el Map-Reply tiene una ruta al Locator en el Locator data record
17. Loc-AFI: un entero de 16 bits que indica la AFI del locator
18. Locator: un valor de 32 o 128 bits que indica la @IP del locator

OOR debe ser capaz de leer este formato de respuestas e interpretarlo adecuadamante, mientras sigue funcionando de la misma forma para el resto de funcionalidades.

Otro requisito es que el prototipo sea capaz de leer transacciones especificadas por el usuario en un fichero de texto. El formato es de libre elección, pero se debe poder ejecutar una función que lea el fichero y "traduzca" esas transacciones a transacciones que sean entendibles por el resto de nodos que estén conectados a la BlockChain.

10.2 Principales decisiones

Para la realización del proyecto se han tomado algunas decisiones relevantes que es interesante comentar. Principalmente hay 3:

10.2.1 ZeroMQ

OOR utiliza ZeroMQ, una librería para comunicación asíncrona. ZeroMQ permite tener sockets TCP, pero tal y com se ha explicado en el apartado anterior, un requisito es que se deben utilizar sockets UDP. ZeroMQ no los soporta, por lo que se ha decidido utilizar sockets normales y no depender de ésta librería.

10.2.2 Pytricia y Radix

Para implementar la funcionalidad de poder resolver la petición de resolución, se necesita una estructura que vincule una dirección IP a una cuenta de BlockChain. Esta estructura, por este requisito, es un árbol Patricia cuyas hojas son las cuentas de BlockChain. Una librería que implementa esta funcionalidad es Pytricia.

En los primeros días de Enero se encontró con información que indicaba que Pytricia tenía un bug no detectado previamente, por lo que, para garantizar el correcto funcionamiento del prototipo, se cambió esta librería a la librería Radix.

10.2.3 Clases Response y BitArray

Para enviar los datos del prototipo a OOR se utiliza la clase *Response* (archivo *map_reply.py*). Esta clase utiliza la clase *BitArray* para crear un bitstream, una tira de bits, que después se convierten en bytes. Esto se ha decidido para poder enviar fácilmente los datos mediante los sockets, ya que éstos solo pueden recibir strings o bytes por parámetro.
Además, utilizar la clase BitArray permite definir las longitudes que cada tira de bits tendrá, por lo que es una clase útil para poder cumplir con los estándares definidos en los requisitos para las respuestas.

11. Modelo de datos

A continuación se explicarán las principales estructuras de datos y clases utilizadas para gestionar las llamadas y las resultados.

11.1 Response

La clase R*esponse* (*map_reply.py*) sirve para poder gestionar los datos de las respuestas que el prototipo genera a raíz de las peticiones de OOR. Para conseguir cumplir con la estructura que se especifica en el apartado 10.1, se utilizan las clases *MapServers* y *MapReplyRecord*. La primera se utiliza para la lista de map servers y la segunda para el map reply (que sigue el modelo especificado en el apartado 6.1.4 del RFC 6380).
Estas clases se incluyen en una variable de la clase *Reply*, creando una estructura anidada.
Mediante esta estructura se consigue tener una clase que puede enviarse muy fácilmente a través de un socket, ya que simplemente es necesario llamar a la función *to_bytes()*, y ya se tendrán los bytes generados.

11.2 OOR

La clase *OOR* (*oor.py)* se utiliza para establecer la conexión con OOR y para recibir y enviar los mensajes. Se utiliza esta clase para poder recibir fácilmente las tres variables que tiene cada petición: nonce, afi y dirección mediante una función. Esto facilita que desde el main se puedan conseguir las peticiones que vayan llegando al socket ocultando el proceso (que puede ser algo engorroso debido al tamaño variable de las direcciones dependiendo de la afi).

11.3 Parser

La clase Parser (*user.py*) se utiliza para traducir las transacciones guardadas en un fichero de texto a transacciones entendibles por la BlockChain.
Mediante la función *read_transactions()* se leen todas las transacciones que contiene el fichero, y después se pueden ir pidiendo una a una mediante la función *get_tx()*.
Esta clase permite, además, que se pueda definir con qué cuenta de las que se dispone se quiere firmar cada una de las transacciones que se crean.
En la BlockChain del prototipo cada cuenta tiene asociada una dirección pública obtenida a través de la clave pública asociada a la cuenta.

11.4 Patricia_state

La clase PatriciaState (*patricia_state.py*) se utiliza para implementar el árbol Patricia exlpicado en el apartado 10.2.2. Utilizar esta clase tiene la ventaja de que permite tener una función para guardar el contenido del árbol en disco y así poder tener persitencia, algo que no ofrece la clase radix.

11.5 Bc_hdr_msg

El struct *bc_hdr_msg* se utiliza para guardar el header de las respuestas que el prototipo envía a OOR. Este struct tiene tres variables, *upper_nonce*, *lower_nonce* y *flag*. El struct divide incorpora la funcionalidad upper y lower del nonce ya que no se pueden coger los 64 bits que ocupa en una llamada y hay que cogerlo de 32 en 32 bits. Además, el struct comprueba si el host en el que se está ejecutando el código utiliza little endian o big endian para saber en qué orden debe coger los parámetros, y el atributo *__packed__* para asegurar el alineamiento.

12. Implementación

A continuación se explicará la implementación de las funciones principales.

12.1 Modificación del código de OOR

El código de OOR puede encontrarse en GitHub[9]. Albert López, responsable de OOR, hizo un fork y me compartió el enlace. En su modificación, dejó anotaciones en la clase lisp_mr, donde indicaba cómo utilizar los componentes de OOR para realizar la conexión entre él mismo y el prototipo.

Casi todo el trabajo se ha realizado en el archivo *lisp_mr.c*. Éste contiene las funciones principales para la recepción de mensajes provenientes de Python. Una de las funciones más importantes que contiene es *mr_ctrl_construct()*, que se encarga de abrir los sockets de escritura y lectura mediante las funciones *open_udp_datagram_socket()*.

[9] https://github.com/OpenOverlayRouter/oor

Tras la apertura de los sockets, se registra el listener en el smaster, que es el módulo maestro y del que cuelgan el resto de módulos de OOR. Esto se consigue con la llamada *sockmstr_register_read_listener*(), función que, entre otros parámetros, recibe la función que se ejecutará cuando se reciba un mensaje por ese socket (en este caso, la función que se registrará será *process_blockchain_api_msg()*).

La función *process_blockchain_api_msg()* se encargará de procesar cada mensaje recibido desde Python. Estos mensajes serán las respuestas a las peticiones de LISP.

Esta función llena un struct *lbuff* (usado en OOR para guardar ciertos datos necesarios) con el contenido del socket. Tras esto, se guardan el nonce y el flag de la respuesta en un struct *bc_hdr_msg* (explicado en el apartado 11.5).

Tras esto, se vacían los datos leídos del buffer para que el puntero se mueva. Estos buffers se leen mediante este puntero, y ha de ir moviéndose para que la lectura sea correcta.
Ahora hay que mirar el flag, que nos dirá si el resto del contenido que hay en el *lbuf* es una lista de map resolvers (flag == 0) o un map reply (flag == 1).

Si el flag es un 0, habrá que parsear los datos del rloc, que serán los siguientes que vendrán.
Además, habrá que obtener el timer asociado a la petición mediante el nonce. En OOR, cada petición que se envíe al módulo de Python tiene un timer asociado a un nonce, que se crea en el momento de enviar. El timer se utiliza para tener datos extras en un struct y para tener un *timeout* (en este caso, 5 segundos), tras el que se considera que no se puede obtener una respuesta a la petición y se muestra un mensaje de error.
Una vez obtenido el timer, se obtienen los datos asociados a la petición que contiene y se utilizan para llamar a la función *build_and_send_encap_map_request()*. Esta función se utiliza para enviar una petición de OOR a una de las direcciones IP contenidas en la lista de map resolvers recibida por el socket. En este caso, lo que se quiere es enviar una petición OOR a esa dirección para que esa dirección sea la que la resuelva.
Se usa la primera dirección de la lista para hacer este envío. Se ha decidido hacer esto por conveniencia (da igual la posición de la lista a la que se envíe, solo importa enviarlo a uno).

Si por el contrario el flag es un 1, se crea un struct *locator_t* y se utiliza la función *lisp_msg_parse_mapping_record()*. Como en este caso la respuesta cumple con el estándar RFC 6380, se puede usar esta función para leer todos los datos “del tirón”, sin necesidad de hacer nada más.
Tras esto se obtiene el timer mediante el nonce (similar al caso de flag 0, esto se hace para obtener los datos asociados a esa petición y poder saber qué dirección IP la ha enviado y enviarle la respuesta generada con los datos obtenidos).

Tanto en los casos del flag 0 y 1, tras obtener los datos y resolver la petición correctamente, se para el timer para que no salte.

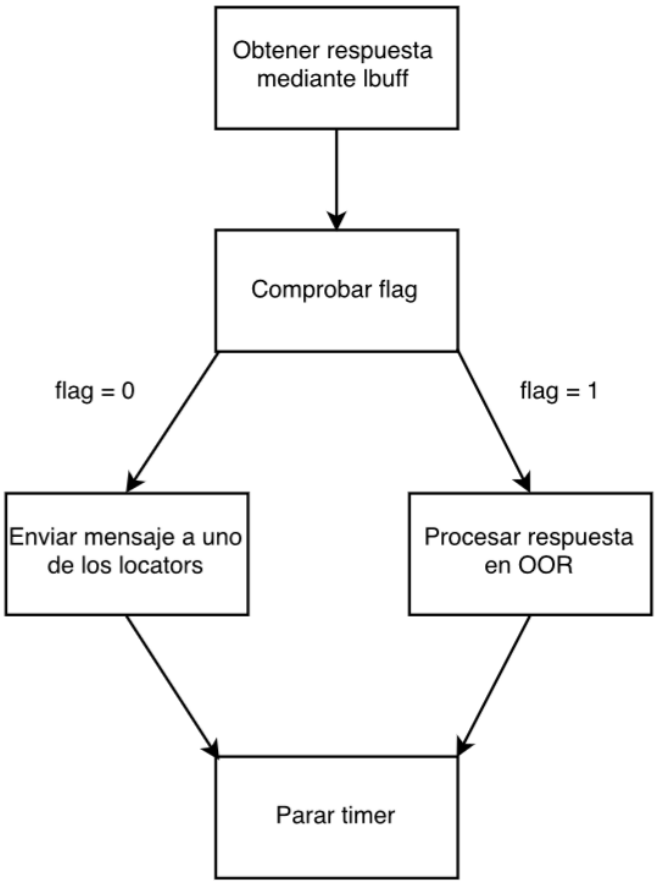


Figura 4: Diagrama de la función *process_blockchain_api_msg()*

La siguiente función a comentar es la *mr_recv_map_request()*. Esta función se ejecuta cuando OOR recibe una petición de resolución de una dirección. La función coge primero el contenido del header que se ha recibido en la petición. Entre estos datos, están el seid (dirección origen) y la dirección a consultar. Tras esto, se envía un mensaje, creando una nueva línea de cache con el timer asociado a la petición (similar al caso visto anteriormente). Para iniciar el timer se debe definir la función que se ejecutará cuando salte, el nonce asociado al mismo, etcétera.
En este timer se guardará la información necesaria para poder procesar la respuesta recibida por el módulo de Python y devolver el resultado al host que ha enviado este map request.
Una vez definidos estos datos, se puede inicializar el timer, indicando un tiempo máximo de espera (el *timeout*). Ahora se crea el mensaje que se enviará a la BlockChain, dando valor a los campos necesarios para la petición (que son el nonce, la afi y la dirección).
Ahora ya se puede enviar el mensaje mediante la función *send_datagram_packet()*, dando por finalizada la función *mr_recv_map_request()*.

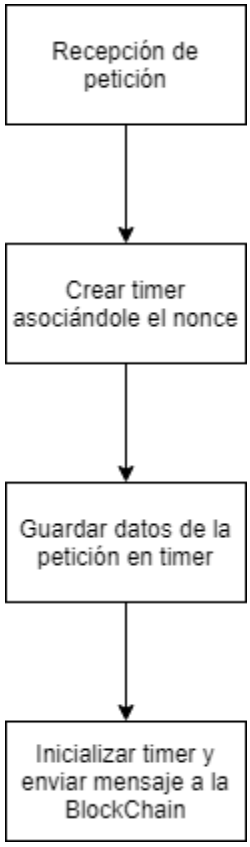


Figura 5: Diagrama de la función mr_recv_map_request()

Tal y como se ha dicho, en el momento en el que se sobrepasa el tiempo de *timeout* del timer, se ejecuta una función. Esta función es *send_map_request_bc()*, que para el timer y borra la línea de cache asociada. Además, se imprime por pantalla el mensaje "Requested time out. Deleting timers..." para informar al usuario de que el timer ha saltado sin que se haya recibido respuesta.

12.2 Main del prototipo

Para conectar los módulos de las 4 partes del proyecto explicadas anteriormente, hice un archivo main (blockchaincba.py). En este archivo se instancian las clases principales de cada una de estas cuatro partes mediante las funciones *init_chain(), init_p2p(), init_consensus(), init_keystore(), init_user()* e *init_oor()*.
Tras esto, se ejecuta un bucle inifinito (*while (not end)*), dentro del que se realizan las 4 funciones prinipales del prototipo:

1. Procesar nuevos bloques
2. Procesar nuevas transacciones provenientes de la red
3. Comprobar si el nodo tiene que firmar el siguiente bloque
4. Procesar nuevas transacciones del usuario
5. Responder queries de OOR
6. Responder queries de la red
7. Responder a la transaction pool

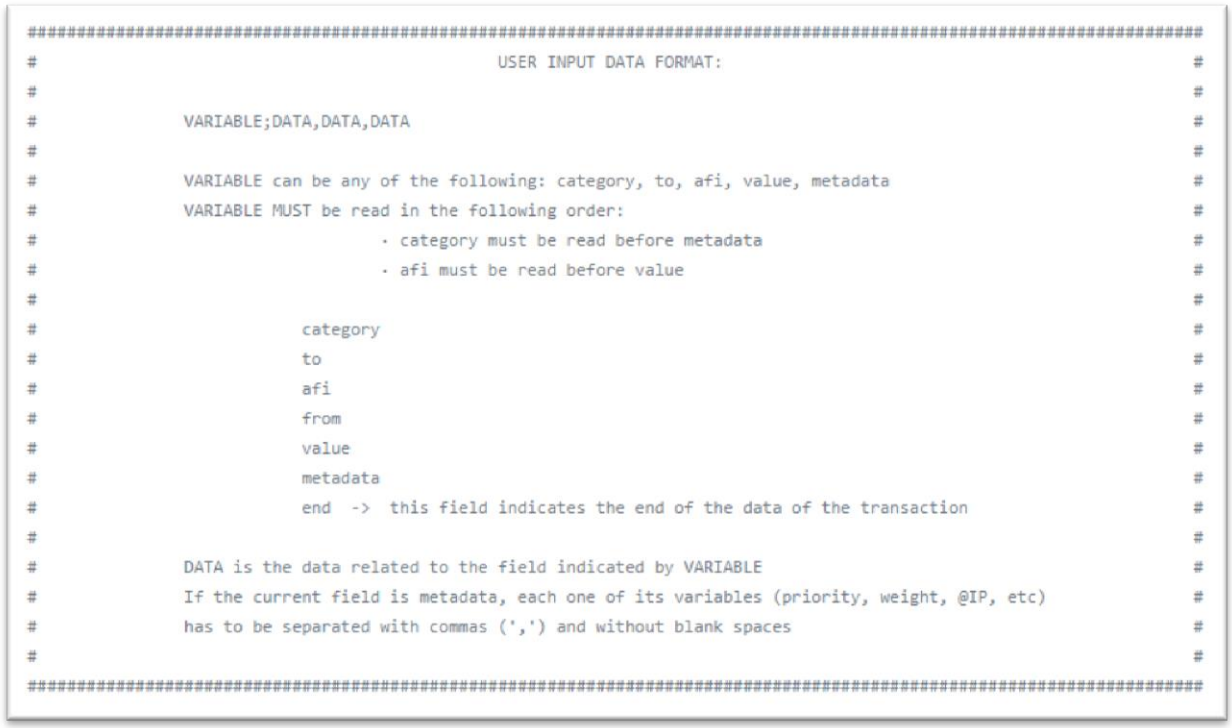
Para las funciones nº 1, 2, 6 y 7 se utiliza el módulo p2p (encargado de la comunicación mediante p2p con el resto de nodos de la red), para la función nº 3 se utiliza el módulo consensus (encargado de gestionar el protocolo de consenso de la BlockChain), para la función nº 4 se utiliza el módulo user (encargado de parsear las transacciones que el usuario haya escrito en un fichero de texto y convertirlas a transacciones que la red BlockChain sea capaz de entender) y para la función nº 5 se utiliza el módulo oor (encargado de la comunicación, mediante sockets, con OOR).

Los mensajes que puedan generar estos módulos se imprimen por terminal mediante la clase logger. Esta clase permite tener un logger que, con un estándar definido por el usuario (en nuestro caso la hora, el nivel de alerta, que puede ser debug, info o critical y el mensaje). Se ha decidido utilizar un logger para facilitar la visibilidad de los distintos tipos de mensajes, separando su importancia por niveles (información del programa, información de debug utilizada durante el proceso de testeo o información generada por excepciones, crítica para el correcto funcionamiento del programa) y con un timestamp para saber exactamente cuándo se ha producido el mensaje.

La necesidad de utilizar un logger quedó patente cuando se unieron todos los módulos y es observó que información generada durante el funcionamiento normal del programa dificultaba ver los posibles mensajes de error generados durante su ejecución.

12.3 Parser de transacciones

Una de las funcionalidades que se esperaba del programa era la capacidad de que el usuario escribiera sus transacciones en un archivo de texto y el prototipo pudiera leerlo y traducirlas para que los nodos que forman la red BlockChain las entendieran. Para conseguirlo se ha creado una clase Parser, contenida en el archivo *user.py*.
En este archivo, se define el formato que debe tener una transacción para que el parser pueda traducirla correctamente. Concretamente, la traducción debe tener el siguiente formato, tal y como se especifica en el archivo:



```
#########################################################################################################################
#                                                USER INPUT DATA FORMAT:                                                #
#                                                                                                                       #
#              VARIABLE;DATA,DATA,DATA                                                                                  #
#                                                                                                                       #
#              VARIABLE can be any of the following: category, to, afi, value, metadata                                 #
#              VARIABLE MUST be read in the following order:                                                            #
#                                  · category must be read before metadata                                              #
#                                  · afi must be read before value                                                      #
#                                                                                                                       #
#                          category                                                                                     #
#                          to                                                                                           #
#                          afi                                                                                          #
#                          from                                                                                         #
#                          value                                                                                        #
#                          metadata                                                                                     #
#                          end  ->  this field indicates the end of the data of the transaction                         #
#                                                                                                                       #
#              DATA is the data related to the field indicated by VARIABLE                                              #
#              If the current field is metadata, each one of its variables (priority, weight, @IP, etc)                #
#              has to be separated with commas (',') and without blank spaces                                           #
#                                                                                                                       #
#########################################################################################################################
```

Figura 6: Formato de las transacciones especificado en el código del archivo user.py

El motivo por el que algunos campos, como el category, han de ser introducidos antes de otros, como el metadata, es debido a que son necesarios para ser usados durante el parseo. En concreto, el campo *category* se necesita antes que el campo *metadata* debido a que la categoría de la transacción modifica el valor de los metadatos de la misma, mientras que el campo *afi* se necesita antes para saber si el campo *value* (que es una dirección IP) es de tipo IPv4 o IPv6 y poder crearla con una clase adecuada a su tipo.

Para diferenciar entre las distintas transacciones guardadas en el fichero de texto se utiliza el string *end*, que le indica al parser que todos los datos de la transacción que se estaba parseando ya han sido leídos. Cuando el parser detecta este string comprueba que la transacción tenga todos los datos que esperaba (es decir, todos los campos especificados en el recuadro anterior) y, si los tiene, los usa para crear una instancia de la clase Transaction, que guarda en un array.

Para obtener una transacción de este array se utiliza la función *get_tx()*.
Se ha decidido utilizar este array y la función *get_tx()* para poder traducir múltiples transacciones de una vez pero no ocasionar problemas al main, que espera poder utilizar las transacciones leídas de una en una. Esto se debe a que, hasta que la transacción no se haya enviado al resto de nodos y ésta se haya añadido al bloque mediante el algoritmo de consenso, dicha transacción no será correcta y, por tanto, la siguiente transacción del usuario leída del fichero de texto no será correcta.
Nunca podrán haber dos transacciones de un mismo usuario en un bloque debido al nonce, un número que indica el número de transacciones que el usuario ha ejecutado.

Si se crean dos transacciones, la primera tendrá un nonce = último nonce + 1, y la segunda tendrá un nonce = último nonce + 2. Cualquier nodo de la red esperará que dicho usuario ejecute una transacción con nonce = último nonce + 1, por lo que la transacción con nonce = último nonce + 2 se rechazará (tiene un nonce inválido) y no se añadirá al bloque.
Por tanto, debido a esto, tener un método de recibir las transacciones leídas de una en una es necesario para el correcto funcionamiento del prototipo.

12.4 Comunicación con OOR

Para la comunicación con OOR se utiliza la clase Oor que se encuentra en el fichero *oor.py*.
Tal y como se ha explicado anteriormente, el prototipo se ha de comunicar con OOR mediante sockets UDP por los puertos 16001 (para enviar mensajes a OOR) y 16002 (para recibir mensajes desde OOR).
Esto se ha hecho en esta clase separada para poder ofrecer funciones que lean las peticiones de OOR (*read_socket()*) y para escribir las respuestas a OOR (*write_socket(answer)*).

Las peticiones de OOR siempre tendrán los siguientes campos:

1. Nonce: un entero de 64 bits
2. AFI: un entero de 16 bits que indica si la dirección es IPv4 (afi = 1) o IPv6 (afi = 2)
3. @IP: la dirección de la que se pide la resolución

La función *read_socket()* se encargará de leer los bytes necesarios del socket para realizar esta lectura correctamente y devolverá los datos de la petición en tres campos diferentes. En caso de que el socket UDP esté vacío se devolverán None como valores para cada uno de los tres campos, de forma que desde el main se pueda detectar que el socket está vacío.

La función *write_socket(answer)* escribirá la respuesta por el socket 16002, por lo que la respuesta ha de tener el formato correcto para que OOR pueda interpretarla. El formato que puede tener la respuesta varía en función de si la respuesta es una lista de map servers o un map reply.

La respuesta siempre tendrá el siguiente formato:

1. Nonce: el nonce que se ha recibido en la petición
2. Flag: un entero de 1 bit que indica si la respuesta es una lista de map servers (flag = 0) o un map reply (flag = 1)
3. Información: puede ser una lista de map servers o un map reply

En el caso de ser una lista de map servers, la respuesta tendrá el siguiente formato:

1. Map server count: un entero de 8 bits que dice el número de map servers que incluirá la respuesta
2. Map server i: información del map server iesimo, que incluye:
   2.1. AFI: entero de 16 bits que indica si la @IP del map server es IPv4 o IPv6

2.2. @IP: la dirección IP del map server, cuyo tamaño varía entre 32 bits (si es IPv4) y 128 bits (si es IPv6)

En el caso de que la información sea un map reply, la respuesta tendrá el siguiente formato, que sigue el estándar definido en la sección 6.1.4 del RFC 6380[10]:

1. TTL: un entero de 32 bits que indica el tiempo, en minutos, que el receptor del map reply guardará la traducción en cache
2. Locator count: un entero de 8 bits que indica el número de locators que contiene el map reply
3. EID mask-len: un entero de 8 bits que indica la longitud de la máscara del EID-prefix
4. ACT: un entero de 3 bits que define acciones del map reply. En concreto, puede haber 4 acciones:
5. No-Action: el map-cache se mantiene vivo y no ocurre encapsulación
6. Natively-Forward: el paquete es natively-forwarded
7. Send-Map-Request: el paquete invoca enviar un Map-Request
8. Drop: un paquete que empareja esta entrada map-cache se dropea. Un mensaje ICMP Destination Unreachable debería ser enviado.
9. Authoritative: un bit que siempre se pone a 1 cuando es enviado por un ETR. Cuando un Map-Server está proxy Map-Replying el bit se pone a 0.
10. Map-Version number: un entero de 12 bits que, cuando es diferente de 0, sirve para que el Map-Reply sender informe al ITR de su número de versión para el record EID del mensaje. Cuando éste vale 0, no hay información de la versión.
11. EID-prefix-AFI: un entero de 16 bits que indica el AFI del EID
12. EID-prefix: un valor de 32 o 128 bits que indica la @IP y cuyo tamaño depende del AFI
13. Priority: un entero de 8 bits que indica la prioridad del locator. Un valor más pequeño indica más prioridad
14. Weight: un entero de 8 bits que indica cómo balancear el tráfico entre RLOCs con el mismo nivel de prioridad
15. M Priority: un entero de 8 bits que define la prioridad multicast del RLOC
16. M Weight: un entero de 8 bits que indica cómo balancear tráfico multicast entre RLOCs con el mismo nivel de prioridad
17. Unused Flags: un valor de 13 bits que se dejan a 0 y se ignoran
18. L: un bit que, cuando se activa, marca el Locator como un local Locator al ETR.
19. p: un bit que, cuando se activa, provoca que el ETR informe al RLOC-probing ITR que la dirección del locator para el que este bit se activa es el que se está RLOC-probing
20. R: un bit que se activa cuando el nodo que envía el Map-Reply tiene una ruta al Locator en el Locator data record
21. Loc-AFI: un entero de 16 bits que indica la AFI del locator
22. Locator: un valor de 32 o 128 bits que indica la @IP del locator

[10] https://datatracker.ietf.org/doc/rfc6830/

Para crear la respuesta se utiliza la clase Response del archivo *map_reply.py*. Esta clase tiene el formato de la respuesta, y su tercer valor (Información) es una instancia de la clase MapServers o de la clase MapReplyRecord.

Para enviar los datos de estas clases a OOR se utiliza la clase BitArray de Python, que crea un bitstream de una longitud predefinida con un valor que reciba por parámetro. Este bistream se convierte a bytes, y son estos bytes los que se envían por el socket. Los motivos por los que se ha tomado esta decisión son los siguientes:

1. En Python se envían strings o bytes por los sockets, por los que no se podrían enviar las clases "tal cual"
2. La clase BitArray permite definir una longitud específica para cada valor a enviar. Esto permite que, por ejemplo, se pueda enviar un entero con un tamaño de 6 bits, pese a que no sea su longitud
3. El bitstream creado por la clase BitArray puede convertirse en bytes fácilmente mediante la función *bytes* de Python

El formato del map reply, que sigue el RFC 6380, permite que en OOR se puedan convertir los bytes recibidos por el socket en una clase de C con una función, sin necesidad de hacer nada más. En el caso de la lista de map servers se debe coger el contador por un lado y las direcciones por otro.

12.5 Patricia_state

La clase Patricia_state se ha explicado en el apartado 11.4, y su implementación no representa una gran dificultad. Aún así, es interesante comentar las funciones *to_db()* y *from_db()*. Estas funciones implementan las propiedades de persistencia, y utilizan Pickle para poder guardar la instancia de la clase en un archivo Patricia.p
En el momento de cargar la base de datos de la BlockChain se carga este árbol Patricia, y con cada modificación se vuelve a guardar en disco para que los cambios sean persistentes.

11. Resultados

Una vez obtenido un prototipo funcional se deben valorar los resultados obtenidos con su funcionamiento.
Cada uno de los miembros del grupo ha realizado pruebas sobre el rendimiento del módulo del que se ha hecho cargo, por lo que en esta memoria se analizará el rendimiento de la comunicación entre OOR y el prototipo (módulo LISP).
Para las pruebas, se medirá el tiempo que se tarda entre el envío de una petición de resolución a OOR y su respuesta.
Estas pruebas se realizarán en un equipo con las siguientes características:

- Procesador: Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz
- RAM: 8094MB
- OS: Ubuntu 16.04.3 LTS
- HDD: TOSHIBA External USB 3.0

Se han realizado un total de 30 peticiones, pidiendo resolver siempre la misma IP (ya que esto no afecta a los resultados).
El tiempo mínimo de resolución ha sido de 0,045 segundos, el tiempo máximo ha sido de 0,498 segundos, y el tiempo medio ha sido de 0,2887 segundos.
Estos resultados permiten observar el funcionamiento esperado en una comunicación mediante sockets UDP y en la que no se observa un retraso muy grande durante el procesamiento de la petición en el prototipo y el cálculo de la respuesta.

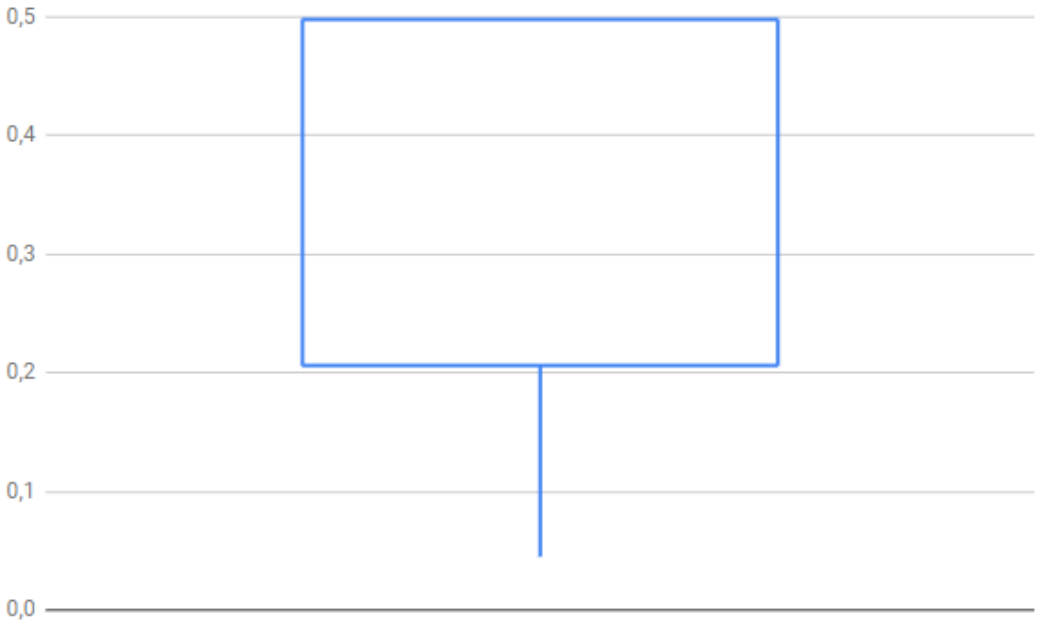


Figura 7: Boxplot de los datos obtenidos en 30 peticiones entre OOR y el prototipo

12. Conclusiones

Se ha conseguido un prototipo completamente funcional que permite implementar la resolución de direcciones LISP sin utilizar certificados para mantener la seguridad del DDT y utilizando en su lugar las propiedades de la BlockChain para obtener un nivel de seguridad similar.
Esto puede ser una motivación para que las entidades responsables de Internet se planteen el uso de LISP, ahora que se puede prescindir de las Autoridades Certificadoras.

Además se ha conseguido un protocolo de consenso Proof of Stake, que permite tener un impacto energético mucho menor que el que podría ofrecer un protocolo Proof of Work. Con esto se ha conseguido mitigar el posible inconveniente que tiene utilizar una BlockChain: el alto consumo de recursos energéticos y la necesidad de Hardware de alto rendimiento.

La adaptación de OOR para que pueda utilizar el prototipo puede conseguir que adaptar esta solución sea muy fácil, debido a que el funcionamiento de OOR seguirá siendo el mismo y el cambio no será detectable para el usuario. De esta forma, se

elimina cualquier posible necesidad de reaprendizaje para adaptarse a esta nueva solución.

Para las pruebas del prototipo completo (no solo el módulo LISP expuesto en esta memoria) se han usado máquinas virtuales situadas en las siguientes localizaciones geográficas:

- Californa
- Virginia
- Frankfurt
- Irlanda
- Sidney
- Tokio
- Barcelona

Y se han realizado pruebas con miles de transacciones, por lo que se puede garantizar que el prototipo escala bien para un ambiente de producción no muy grande.

Por tanto, se han cumplido todos los objetivos propuestos en el apartado 3.1 siguiendo la metodología de trabajo explicada en el apartado 4, por lo que se considera que se ha cumplido el proyecto propuesto satisfactoriamente.

13. Trabajo futuro

Pese a haber conseguido los objetivos propuestos, sería deseable seguir trabajando en el prototipo para poder mejorar algunas funcionalidades realizadas y poder añadir algunas nuevas como, por ejemplo, probar el prototipo en un ambiente más “realista” en el que haya más máquinas virtuales y un número de transacciones aún más grandes y sostenido en el tiempo.

También sería deseable realizar una documentación extensa de los diferentes módulos para poder facilitar la comprensión del código, su funcionamiento y su posible adaptación para propósitos distintos.

Un sistema de logs persistentes (y no en consola) sería también deseable para poder recrear bugs que se produjeran durante el funcionamiento no supervisado del prototipo, así como un sistema de alertas (mediante correos electrónicos, por ejemplo) en caso de que tal hecho tuviera lugar.

14. Agradecimientos

Este trabajo no habría sido posible sin la ayuda del director del TFG Albert Cabellos ni sin la ayuda de Jordi Paillissé, quienes guiaron al equipo durante la realización de todo el proyecto y aportaron ideas que permitieron completarlo.

Albert López, encargado de OOR, aportó el conocimiento necesario sobre el funcionamiento de OOR sin el cual habría sido mucho más difícil la modificación exitosa del mismo.

Bibliografía

[1] CABELLOS, A., PAILLISSE, J. et. al. (2017). "Programmable Overlays via OpenOverlayRouter". *IEEE Communications Magazine*, volume 55, issue 6: 32-38.

[2] CABELLOS, A. et. al. (2012). "An Architectural Introduction to the Locator/ID Separation Protocol (LISP)". *IETF Datatracker*.

[3] IEEE-USA. (2009). "Next Generation Internet: IPv4 Address Exhaustion, Mitigation Strategies and Implications for the U.S".

[4] FULLER, V et. al. (2011). "LISP Delegated Database Tree". *IETF Datatracker.*

[5] FARINACCI, D. et. al. (2009) "Locator/ID Separation Protocol (LISP)". *IETF Datatracker.*

[6] IANNONE, L. y BONAVENTURE, O. (2007) "Locator/ID Separation: Study on the costs of Mappings Caching and Mapping Lookups".

[7] CROSBY, M et. al. (2015). "BlockChain Technology, Beyond Bitcoin".

[8] SILVA, J. E. (2003). "An Overview of Cryptographic Hash Functions and Their Uses". *GIAC Security Essentials Practical.*

[9] NAKAMOTO, S. (2008). "Bitcoin: A Peer-to-Peer Electronic Cash System"

[10] BASTIAAN, M. (2015). "Preventing the 51%-Attack: a Stochastic Analysis of Two Phase Proof of Work in Bitcoin".

[11] WOOD, G. (2012). "Ethereum: A Secure Decentralised Generalised Transaction Ledger (EIP-150 Revision)".

[12] O'DWYER, K.J, MALONE, D. (2014). "Bitcoin Mining and its Energy Footprint". *IET Irish Signals & Systems Conference 2014*.

[13] KING, S., NADAL, S. (2012). "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake".

[14] BITFURY GROUP. (2015). "Proof of Stake versus Proof of Work".

[15] DEMEESTER, T. (2017). "Critique of Buterin's "A Proof of Stake Design Philosophy"".

[16] BUTERIN, V., GRIFFITH, V. (2017). “Casper the Friendly Finality Gadget”.

Fuentes de las figuras

Figura 1. https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/locator-id-separation-protocol-lisp/datasheet_c78-576698.html

Figura 2. https://danims.com/wp-content/uploads/2015/06/bitcoin-block-chain-verified.png

Figura 3. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ

Figura 4. Elaboración propia

Figura 5. Elaboración propia

Figura 6. Elaboración propia

Figura 7. Elaboración propia